Homework Assignment 6

You are to read a simplified version of internet packet data and reassemble the packets into sorted order so that the underlying message can be read.

An individual "packet" will consist of a single line of input data containing a `messageID`, a `packetID`, a `packetCount`, and a `payload`. The first three of these are to be `int` variables and the last will be a `String`. The `Packet` class is essentially the same thing as what we have been calling a `Record` class all through the semester, and an `interface IPacket` can be found on the website.

For ease in figuring out where to do what, the `Driver` class is also on the website. The `Driver` class is identical or nearly identical to the same class of a previous exercise.

The work of the assembly of packets is to be done in a `PacketAssembler` class for which an `interface IPacketAssembler` is on the website.

In the previous exercise, you had to do your own insertion sort to keep the incoming packets in correct sorted order by packet ID number. For this assignment, you are to use a `TreeMap` to keep them in order as they come into the system dynamically.

The `PacketAssembler` will have a `TreeMap` of `Message` values indexed by the `Integer` values that are the `messageID`s.

The `Message` will have a `TreeMap` of `Packet` values indexed by the `Integer` values that are the `packetID`s.

Data that is global to the message will also have an instance variable in the `Message` class.

In other words, the packets will come in in no known order, but by storing them in a `TreeMap`, you will avoid the need to actually sort based on `packetID`. The search tree you create will allow you to read the packets back in sorted order by `packetID` simply by traversing the tree in keyset order.

In addition, the messages will come in in no known order, but the order by message ID will be kept by the `TreeMap` in the `PacketAssembler`.

When a message is complete, it is to be printed to the output file and deleted from the `TreeMap`.

Three sample input data files can be found on the website together with the output that could/should go along with the input. Note that the "scrambled" versions are the ones in a randomized order. You should for the purposes of this exercise print out all four of the instance variables of the `Packet` class. In a more real application, you might only need to reassemble the `payload`, but for debugging purposes it's a good idea to print out all the information just to make sure you got it right by design and not by serendipity.

Your code must handle duplicate instances of packets (duplicate by ID) and duplicate instances of messages. That is, if the complete set of packets for a given message comes in after the message has been completed, printed, and deleted, then your code should do all the work all over again, because it won't have saved any information to know that it has already seen that message.